Workers Network

Final project
Design and software system development
Internet web service
2021-2022

Student's name: Ali Tarek Abusaleh.
High-school: Jamal Tarabeh – Al-Hekma.
ID number: [illegible]93.
Teacher's name: Hanan Darawsha.

Table of contents:

Introduction

This year, my classmates and I were told that the subject of our project, must be about sustainability. We had to think about sustainability all the time, and that in return helped us develop a deeper understanding of our future.

Now you may ask, "What is your project about? And how is it about sustainability?".

To answer your questions, first I need to introduce my project, what its uses are, and how does it help.

Project introduction:

In my project, there are two types of users other than the admin. The two users benefit from each other and my website makes life easier for both.

The two users are: 1. Workers. 2. Customers.

1. Workers:
   Workers get a chance to access more work opportunities from my website.
   What they have to do is: they have to sign-up and make an account. In this account they can type their professions, experience years, a biograph, and much more!

They can get orders from customers and can access them through the website, then they can decide whether to deny or accept the order.

2. Customers:
   As expected, customers also get a chance to access workers to get the job done for them using my website. They also can choose which worker to order a work from, using the data that the worker provided, or using the ratings of others.
   What a customer has to do is sign-up. The unique thing here is that customers have to provide their address/location, they can just provide the city name or the area if they do not want to be exact, and then the worker can contact them through their emails or phone-numbers.
   Customers can also access their orders. They can remove most orders, other than the ones with the status "Accepted". This is to ensure that workers' schedules don't get missed with.

As stated before, customers can choose which worker they want to order from. How can they? The website is provided with the ability to search for workers using their tags. Workers get to edit their tags in their profile settings page. Using the search bar that is accessible for both users, they can enter profession names or job names, after that the

search engine examines the words that have been entered and compares them to all workers-tags. This is a reason that makes the PREFERED way to search is using only one word. This will insure best results.
All users including the admins get a guide in their homepage to help them understand how the website works.
The first intention for customers home page was to recommend other workers to them in the homepage based on their search history. I had it all planned but the lack of time did not help, so I provided only guides in the homepage.

This is connected to sustainability because it sustains work opportunities for workers. In our time, a lot of people have talents or degrees that allow them to be helpful in today's society. But they don’t get a chance to work or provide because of the lack of work opportunities. My website works to fix this problem.

Tables

tblAdmins

this table contains the data about the admins that have been added to the website by other admins.

Admin id (automatic number), name, email address, username, password.
The admin logs-in using email and password.

Field Name	Data Type	
adminID	AutoNumber	A unique serial number for the admin
firstName	Short Text	the admin's first name
lastName	Short Text	the admin's last name
email	Short Text	the admin's email address
userName	Short Text	the admin's username
password	Short Text	the admin's password to their account

tblCustomers

this table contains the data about the customers that they have entered in the sign-up, or that they have changed in the profile settings.

Customer id, name, birth date, phone-number, email address, username, password, address description.
The customer logs-in using email and password.

Field Name	Data Type	
customerID	Short Text	the customer's ID number
firstName	Short Text	the customer's first name
lastName	Short Text	the customer's last name
dateOfBirth	Date/Time	the customer's birth date
phoneNumber	Short Text	the customer's phone-number
email	Short Text	the customer's email address
userName	Short Text	the customer's username
password	Short Text	the customer's password to their account
addressDescription	Short Text	the customer's address description

tblOrders

this table contains the data about the orders that has been entered by customers when making orders or has been updated by workers.

Order id (automatic number), customer id, worker id, order title, order description, order date, order status.
The customer determines most properties when they make the order. But the status is changed only when workers change it.

Field Name	Data Type	
orderID	AutoNumber	A unique serial number for the admin
customerID	Short Text	the ID number of the customer that ordered
workerID	Short Text	the ID number of the worker that has been ordered
title	Short Text	the title of the order
description	Long Text	the description of the order
orderDate	Date/Time	the date the order has been ordered on
status	Short Text	the status of the order

tblWorkers

this table contains the data about the workers that they have entered in the sign-up, or that they have changed in the profile settings.

Worker id, name, birth date, phone-number, email address, username, password, profession, image URL, biograph, status.
The worker logs-in using email and password.

Field Name	Data Type	
workerID	Short Text	the worker's ID number
firstName	Short Text	the worker's first name
lastName	Short Text	the worker's last name
dateOfBirth	Date/Time	the worker's birth date
phoneNumber	Short Text	the worker's phone-number
email	Short Text	the worker's email address
userName	Short Text	the worker's username
password	Short Text	the worker's password to their account
profession	Short Text	the worker's profession
ImageURL	Short Text	the URL of the image that the worker chose
biograph	Short Text	the worker's biograph
status	Short Text	the worker's status

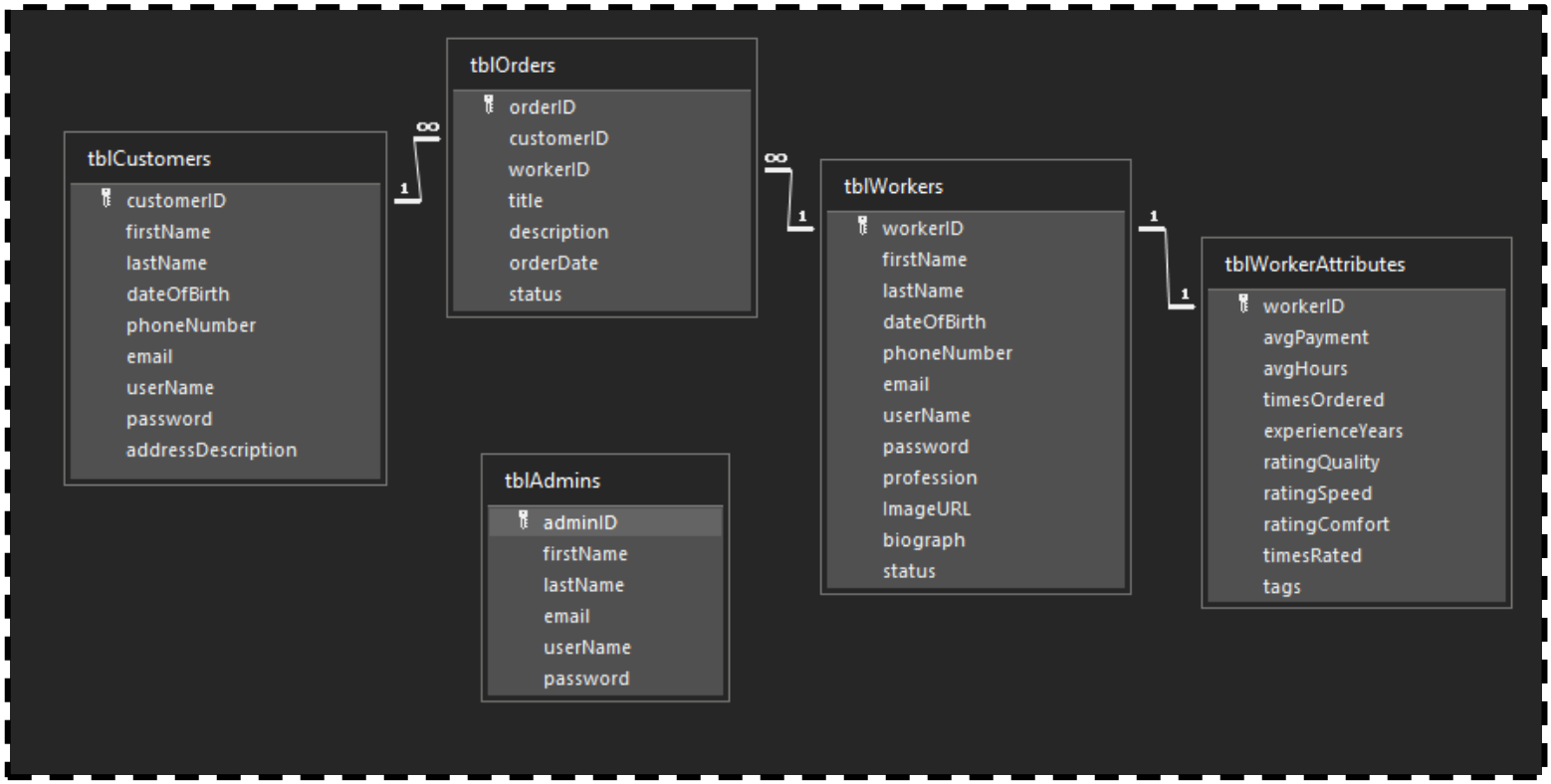
tblWorkerAttributes

this table contains the data about the workers that they have entered in the profile settings. It also includes data that has been set by customers. Like ratings.

Worker id, average payment per hour, average work hours, times ordered number, experience years, quality rating, speed rating, comfort rating, times rated number, tags.
This data is not set in the sign-up other than the id, so I decided to split the workers table into two, for easier use.

Field Name	Data Type	
workerID	Short Text	the worker's ID number
avgPayment	Currency	the worker's average payment for an hour of work
avgHours	Number	the worker's average work hours in a day of work
timesOrdered	Number	the number of times the worker has been ordered
experienceYears	Number	the experience years that the worker has
ratingQuality	Number	the quality rating for the worker provided by customers
ratingSpeed	Number	the speed rating for the worker provided by customers
ratingComfort	Number	the comfort rating for the worker provided by customers
timesRated	Number	the number of times the worker has been rated
tags	Long Text	the tags that the worker provided

Table relationships



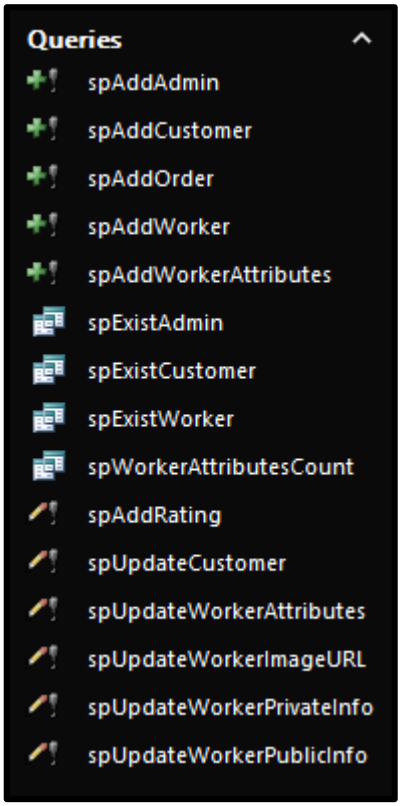
Relationships between tables:

- tblCustomers is connected with tblOrders with one-to-many connection type.
  Meaning that customers can have many orders but not the opposite.

- tblWorkers is connected with tblOrders with one-to-many connection type.
  Meaning that workers can be ordered many times but not the opposite.

- tblWorkerAttributes is connected with tblWorkers with one-to-one connection type. Meaning that every worker has only one workerAttributes row in that table.

Queries



spAddAdmin

INSERT INTO tblAdmins (firstName, lastName, Email, UserName, [Password])

VALUES ([@FirstName], [@LastName], [@Email], [@UserName], [@Password]);

The use of this query is to add an admin. Only other admins can use this query.

spAddCustomer

INSERT INTO tblCustomers (customerID, FirstName, LastName, dateOfBirth, phoneNumber, email, userName, [password], addressDescription)

VALUES ([@CustomerID], [@FirstName], [@LastName], [@DateOfBirth], [@PhoneNumber], [@Email], [@UserName], [@Password], [@AddressDescription]);

The use of this query is to add a customer. This query is only used when a customer signs-up.

spAddOrder

INSERT INTO tblOrders (customerID, workerID, title, description)

VALUES ([@CustomerID], [@WorkerID], [@Title], [@Description]);

The use of this query is to add an order. This query is only used when a customer makes an order.

spAddWorker

INSERT INTO tblWorkers (workerID, firstName, lastName, dateOfBirth, phoneNumber, email, userName, [password], profession)

VALUES ([@WorkerID], [@FirstName], [@LastName], [@DateOfBirth], [@PhoneNumber], [@Email], [@UserName], [@Password], [@Profession]);

The use of this query is to add a worker. This query is only used when a worker signs-up, or when an admin adds a worker.

spAddWorkerAttributes

INSERT INTO tblWorkerAttributes (workerID, tags)

VALUES ([@WorkerID], [@Tags]);

The use of this query is to add a workerAttributes. This query is used automatically when a worker is added. It adds the necessary value - workerID, and adds tags that have been decided in the C# code.

spExistAdmin

SELECT count(*)
FROM tblAdmins
WHERE email=[@Email];

The use of this query is to count the number of admins that exist with the same email that is provided.
It is essentially used to check if an admin with the email provided already exists or not, to avoid duplicates.

spExistCustomer

SELECT count(*)
FROM tblCustomers
WHERE email=[@Email] OR customerID=[@CustomerID];

The use of this query is to count the number of customers that exist with the same email OR the same customerID that is provided.
It is essentially used to check if a customer with the email OR the ID provided already exists or not, to avoid duplicates.

spExistWorker

```
SELECT count(*)
FROM tblWorkers
WHERE email=[@Email] OR workerID=[@WorkerID];
```

The use of this query is to count the number of workers that exist with the same email OR the same customerID that is provided.
It is essentially used to check if a worker with the email OR the ID provided already exists or not, to avoid duplicates.

spWorkerAttributesCount

```
SELECT count(*)
FROM tblWorkerAttributes;
```

The use of this query is to count the number of workerAttributes that exist.

spAddRating

```
UPDATE tblWorkerAttributes SET ratingQuality =
[@RatingQuality], ratingSpeed = [@ratingSpeed],
ratingComfort = [@RatingComfort], timesRated =
[@TimesRated]

WHERE workerID = [@ID];
```

The use of this query is to update the ratings of the worker. It is used when a customer rates a worker.

spUpdateCustomer

UPDATE tblCustomers SET firstName = [@FirstName], lastName = [@LastName], dateOfBirth = [@DateOfBirth], phoneNumber = [@PhoneNumber], userName = [@UserName], email = [@Email], [password] = [@Password], addressDescription = [@AddressDescription]

WHERE customerID = [@ID];

The use of this query is to update everything about the customer. It is used only when a customer changes things in profile settings.

spUpdateWorkerAttributes

UPDATE tblWorkerAttributes SET avgPayment = [@AvgPayment], avgHours = [@AvgHours], experienceYears = [@ExperienceYears], tags = [@Tags], timesOrdered = [@TimesOrdered]

WHERE workerID = [@ID];

The use of this query is to update most things about workerAttributes. It is used only when a worker changes things in profile settings.

spUpdateWorkerImageURL

UPDATE tblWorkers SET imageURL = [@ImageURL]
WHERE workerID = [@ID];

The use of this query is to update the image URL of the worker's PFP. It is used only when a worker updates it in his profile settings

spUpdateWorkerPrivateInfo

UPDATE tblWorkers SET email = [@Email], [password] = [@Password]

WHERE workerID = [@ID];

The use of this query is to update the account information related to the worker. It is used only when a worker changes them in account settings.

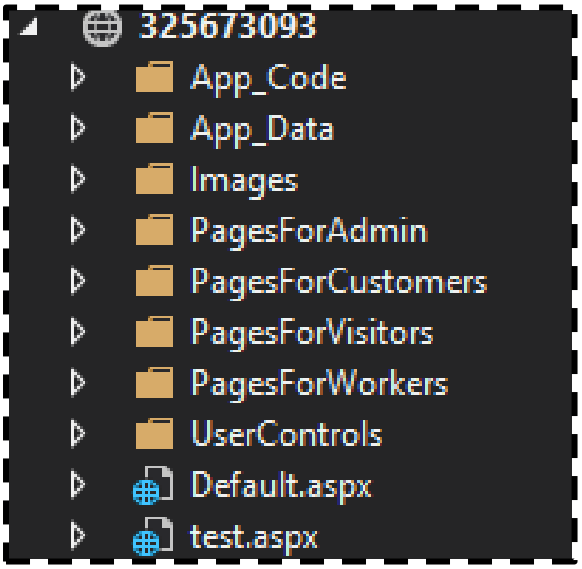
spUpdateWorkerPublicInfo

UPDATE tblWorkers SET firstName = [@FirstName], lastName = [@LastName], dateOfBirth = [@DateOfBirth], phoneNumber = [@PhoneNumber], userName = [@UserName], imageURL = [@ImageURL], biograph = [@Biograph], status = [@Status]

WHERE workerID = [@ID];

The use of this query is to update most things about the worker. It is used only when a worker changes things in profile settings.

Folders Structure



App_Code: a folder that contains all the important classes, contains DoQueries that has functions we use a lot. The other classes are a recreation of the access tables, in a form of classes.

App_data: a folder that contains the file of the access database.

Images: a folder that contains all the images used in the website. It also contains all the PFPs that the workers upload in a folder named "PFPs".

PagesForAdmin: a folder that contains all the webforms that the admin has access to and can use.

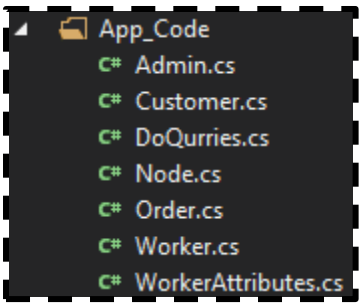
They all rely on the main design of-mpAdmins.master .

PagesForCustomers: a folder that contains all the webforms that a customer has access to and can use. They all rely on the main design of-mpCustomers.master .

PagesForVisitors: a folder that contains all the webforms that a visitor has access to and can use. The visitor is someone who is not logged in. Thus, it only includes the starting page and the login page.

PagesForWorkers: a folder that contains all the webforms that a worker has access to and can use. They all rely on the main design of-mpWorkers.master .

UserControls: a folder that contains all the Web-User-Controls that I have created.
Examples: wucAdmin, wucWorker, wucCustomer. These have been used in their own MasterPage respectively.
I have created other Controls for their repetitive appearance in my website.



App_Code - business logic layer

This folder contains all the classes used in this website. Classes that contain functions only, and the classes that were translated from Access to C#, we split those into two parts, in the first part we build a public class that contains all the properties from the targeted table in Access, and in the second part, we build another public class that contains all the functions that we use on the object from that type. We do that because in Access, tables only contain properties, while the functions are in the Queries section.

For Example: in the class Customer, there are two classes:
CustomerDetails: contains all the properties from tblCustomers, with getters & setters.
Customer: contains all the functions that will be used on the object of type CustomerDetails. Like ExistCustomer…

Admin.cs

```
//**************Class  AdminDetails - begin ***************
public class AdminDetails
{
    //Encapsulation
    private int _AdminID;
    private string _FirstName;
    private string _LastName;
    private string _Email;
    private string _UserName;
    private string _Password;

    //Properties
    public int AdminID
    {
        get { return _AdminID; }
        set { _AdminID = value; }
    }
    public string FirstName
    {
        get { return _FirstName; }
        set { _FirstName = value; }
    }
    public string LastName
    {
        get { return _LastName; }
        set { _LastName = value; }
    }
    public string Email
    {
        get { return _Email; }
        set { _Email = value; }
    }
    public string UserName
    {
        get { return _UserName; }
        set { _UserName = value; }
    }
    public string Password
    {
        get { return _Password; }
        set { _Password = value; }
    }
    public AdminDetails()
    {
    }
}
//**************Class  AdminDetails - end   ***************
//**************Class  Admin - begin **********************
public class Admin
{
    //Encapsulation
    ArrayList prmList;
    OleDbParameter prm;
    string strSQLName;
```

All properties from tblAdmins has been translated to here.

Getters & Setters

Constructor

Properties

```
//Constructor
public Admin()
  {

  }
  //Methods

  //____________________AddAdmin_____________________________

  public void AddAdmin(AdminDetails admDetails)
  {
      strSQLName = "spAddAdmin";
      prmList = new ArrayList();

      prm = new OleDbParameter("@FirstName", OleDbType.VarChar);
      prm.Value = admDetails.FirstName;
      prmList.Add(prm);

      prm = new OleDbParameter("@LastName", OleDbType.VarChar);
      prm.Value = admDetails.LastName;
      prmList.Add(prm);

      prm = new OleDbParameter("@Email", OleDbType.VarChar);
      prm.Value = admDetails.Email;
      prmList.Add(prm);

      prm = new OleDbParameter("@UserName", OleDbType.VarChar);
      prm.Value = admDetails.UserName;
      prmList.Add(prm);

      prm = new OleDbParameter("@Password", OleDbType.VarChar);
      prm.Value = admDetails.Password;
      prmList.Add(prm);

      DoQueries.ExecuteSPNonQuery(strSQLName, prmList);
  }

//____________________ExistAdmin_____________________________

  public Boolean ExistAdmin(AdminDetails admDetails)
  {
      strSQLName = "spExistAdmin";
      prmList = new ArrayList();

      prm = new OleDbParameter("@Email", OleDbType.VarChar);
      prm.Value = admDetails.Email;
      prmList.Add(prm);

      int intCount = (int)DoQueries.ExecuteSPScalar(strSQLName, prmList);
      return intCount > 0;
  }


  public Boolean ExistAdmin(string email)
  {
      strSQLName = "spExistAdmin";
      prmList = new ArrayList();
```

Constructor

This function uses the query in Access to add an admin. It takes AdminDetails as parameter, and outputs nothing.

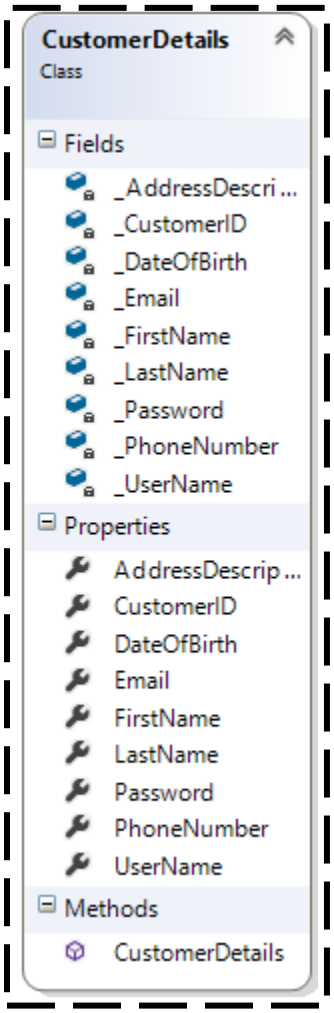
This function uses the query in Access to check if an admin exists. It takes AdminDetails as a parameter, and outputs true/false accordingly.

This function is like the above. But it takes id as a parameter, and outputs true/false accordingly.

```
            prm = new OleDbParameter("@Email", OleDbType.VarChar);
            prm.Value = email;
            prmList.Add(prm);

            int intCount = (int)DoQueries.ExecuteSPScalar(strSQLName, prmList);
            return intCount > 0;
        }

        public AdminDetails GetAdminByEmail(string email)
        {
            AdminDetails e = new AdminDetails();

            if (ExistAdmin(email))
            {
                string strDetails =
                    string.Format("SELECT * FROM tblAdmins" +
                    " WHERE email='{0}'", email);
                DataSet ds = DoQueries.ExecuteDataSet(strDetails);
                e.AdminID = int.Parse(ds.Tables[0].Rows[0]["adminID"].ToString());
                e.FirstName = ds.Tables[0].Rows[0]["firstName"].ToString();
                e.LastName = ds.Tables[0].Rows[0]["lastName"].ToString();
                e.Email = ds.Tables[0].Rows[0]["email"].ToString();
                e.UserName = ds.Tables[0].Rows[0]["userName"].ToString();
                e.Password = ds.Tables[0].Rows[0]["password"].ToString();
            }
            return e;
        }
    }
    //**************Class  Admin - end   **********************
```

This function gets the admin by their email using the SQL syntax that I typed.
It takes email as a parameter, and outputs AdminDetails object.

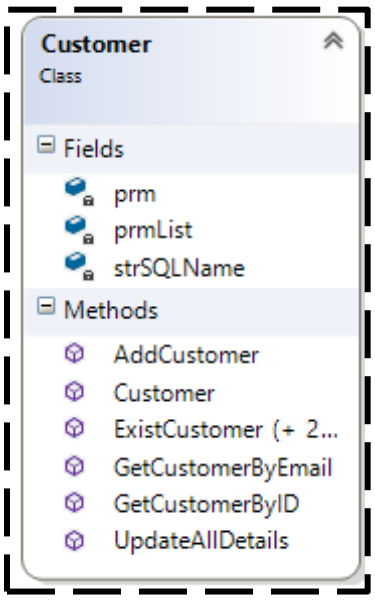
Customer.cs



This class is made to contain fields.
Properties and Fields: this class has the same fields that tblCustomer has. This is so we can access the customer data and modify it.
Methods: there is only one method here, a constructor. This class is not made to contain methods.

```
//**************Class  CustomerDetails - begin ***************
public class CustomerDetails
{
    private string _CustomerID;
    private string _FirstName;
    private string _LastName;
    private DateTime _DateOfBirth;
    private string _Email;
    private string _UserName;
    private string _Password;
    private string _PhoneNumber;
    private string _AddressDescription;


    public CustomerDetails()
    {

    }
```

```
        //Properties
        public string CustomerID
        {
            get { return _CustomerID; }
            set { _CustomerID = value; }
        }
        public string FirstName
        {
            get { return _FirstName; }
            set { _FirstName = value; }
        }
        public string LastName
        {
            get { return _LastName; }
            set { _LastName = value; }
        }
        public DateTime DateOfBirth
        {
            get { return _DateOfBirth; }
            set { _DateOfBirth = value; }
        }
        public string Email
        {
            get { return _Email; }
            set { _Email = value; }
        }
        public string UserName
        {
            get { return _UserName; }
            set { _UserName= value; }
        }
        public string Password
        {
            get { return _Password; }
            set { _Password = value; }
        }
        public string PhoneNumber
        {
            get { return _PhoneNumber; }
            set { _PhoneNumber = value; }
        }
        public string AddressDescription
        {
            get { return _AddressDescription; }
            set { _AddressDescription = value; }
        }


    }
    //**************Class  CustomerDetails - end ***************
```



This class is made to contain methods.
Fields: the fields here are variables that we will use in methods later.
Methods:-
Customer: constructor.
AddCustomer: adds a customer using the query spAddCustomer.
ExistCustomer: checks if the customer exists using the query spExistCustomer.
GetCustomerByEmail: gets the customer by email as CustomerDetails object.
GetCustomerByID: gets the customer by ID as CustomerDetails object.
UpdateAllDetails: updates all customer details using the query spUpdateCustomer.

```
//**************Class  Customer - begin ********************
public class Customer
{
    //Encapsulation
    ArrayList prmList;
    OleDbParameter prm;
    string strSQLName;

    //Constructor
    public Customer()
    {
    }
    //Methods

    //____________________AddCustomer_____________________

    public void AddCustomer(CustomerDetails cstDetails)
    {
        strSQLName = "spAddCustomer";
        prmList = new ArrayList();

        prm = new OleDbParameter("@CustomerID", OleDbType.VarChar);
        prm.Value = cstDetails.CustomerID;
        prmList.Add(prm);

        prm = new OleDbParameter("@FirstName", OleDbType.VarChar);
        prm.Value = cstDetails.FirstName;
        prmList.Add(prm);
```

```
        prm = new OleDbParameter("@LastName", OleDbType.VarChar);
        prm.Value = cstDetails.LastName;
        prmList.Add(prm);

        prm = new OleDbParameter("@DateOfBirth", OleDbType.VarChar);
        prm.Value = cstDetails.DateOfBirth;
        prmList.Add(prm);

        prm = new OleDbParameter("@PhoneNumber", OleDbType.VarChar);
        prm.Value = cstDetails.PhoneNumber;
        prmList.Add(prm);

        prm = new OleDbParameter("@Email", OleDbType.VarChar);
        prm.Value = cstDetails.Email;
        prmList.Add(prm);

        prm = new OleDbParameter("@UserName", OleDbType.VarChar);
        prm.Value = cstDetails.UserName;
        prmList.Add(prm);

        prm = new OleDbParameter("@Password", OleDbType.VarChar);
        prm.Value = cstDetails.Password;
        prmList.Add(prm);

        prm = new OleDbParameter("@AddressDescription", OleDbType.VarChar);
        prm.Value = cstDetails.AddressDescription;
        prmList.Add(prm);


        DoQueries.ExecuteSPNonQuery(strSQLName, prmList);
    }
    //____________________ExistCustomer_____________________________

    public Boolean ExistCustomer(string Email,bool useless)
    {
        strSQLName = "spExistCustomer";
        prmList = new ArrayList();

        prm = new OleDbParameter("@Email", OleDbType.VarChar);
        prm.Value = Email;
        prmList.Add(prm);

        prm = new OleDbParameter("@CustomerID", OleDbType.VarChar);
        prm.Value = "a-1-8-8-8-7-9";
        prmList.Add(prm);

        int intCount = (int)DoQueries.ExecuteSPScalar(strSQLName, prmList);
        return intCount > 0;
    }
    public Boolean ExistCustomer(CustomerDetails cstDetails)
    {
        strSQLName = "spExistCustomer";
        prmList = new ArrayList();

        prm = new OleDbParameter("@Email", OleDbType.VarChar);
        prm.Value = cstDetails.Email;
        prmList.Add(prm);

        prm = new OleDbParameter("@CustomerID", OleDbType.VarChar);
```

```
            prm.Value = "a-1-8-8-8-7-9";
            prmList.Add(prm);

            int intCount = (int)DoQueries.ExecuteSPScalar(strSQLName, prmList);
            return intCount > 0;
        }
        public Boolean ExistCustomer(string ID)
        {
            strSQLName = "spExistCustomer";
            prmList = new ArrayList();

            prm = new OleDbParameter("@Email", OleDbType.VarChar);
            prm.Value ="a-1-8-8-8-7-9";
            prmList.Add(prm);

            prm = new OleDbParameter("@CustomerID", OleDbType.VarChar);
            prm.Value = ID;
            prmList.Add(prm);

            int intCount = (int)DoQueries.ExecuteSPScalar(strSQLName, prmList);
            return intCount > 0;
        }


        public CustomerDetails GetCustomerByID(string id)
        {
            CustomerDetails e = new CustomerDetails();

            if (ExistCustomer(id))
            {
                string strDetails =
                    string.Format("SELECT * FROM tblCustomers" +
                    " WHERE customerID='{0}'", id);
                DataSet ds = DoQueries.ExecuteDataSet(strDetails);
                e.CustomerID = ds.Tables[0].Rows[0]["customerID"].ToString();
                e.FirstName = ds.Tables[0].Rows[0]["firstName"].ToString();
                e.LastName = ds.Tables[0].Rows[0]["lastName"].ToString();
                {
                    int day, month, year;
                    DateTime date;
                    string strDate =
ds.Tables[0].Rows[0]["dateOfBirth"].ToString();

                    month = int.Parse(strDate.Substring(0, strDate.IndexOf("/")));
                    strDate = strDate.Substring(strDate.IndexOf("/") + 1);
                    day = int.Parse(strDate.Substring(0, strDate.IndexOf("/")));
                    strDate = strDate.Substring(strDate.IndexOf("/") + 1);
                    year = int.Parse(strDate.Substring(0, strDate.IndexOf(" ")));

                    date = new DateTime(year, month, day);

                    e.DateOfBirth = date;
                }//getting the date
                e.PhoneNumber = ds.Tables[0].Rows[0]["phoneNumber"].ToString();
                e.Email = ds.Tables[0].Rows[0]["email"].ToString();
                e.UserName = ds.Tables[0].Rows[0]["userName"].ToString();
                e.Password = ds.Tables[0].Rows[0]["password"].ToString();
                e.AddressDescription =
ds.Tables[0].Rows[0]["addressDescription"].ToString();
            }
```

```
            return e;
        }

        public CustomerDetails GetCustomerByEmail(string email)
        {
            CustomerDetails e = new CustomerDetails();

            if (ExistCustomer(email,true))
            {
                string strDetails =
                    string.Format("SELECT * FROM tblCustomers" +
                    " WHERE email='{0}'", email);
                DataSet ds = DoQueries.ExecuteDataSet(strDetails);
                e.CustomerID = ds.Tables[0].Rows[0]["customerID"].ToString();
                e.FirstName = ds.Tables[0].Rows[0]["firstName"].ToString();
                e.LastName = ds.Tables[0].Rows[0]["lastName"].ToString();
                {
                    int day, month, year;
                    DateTime date;
                    string strDate =
    ds.Tables[0].Rows[0]["dateOfBirth"].ToString();

                    month = int.Parse(strDate.Substring(0, strDate.IndexOf("/")));
                    strDate = strDate.Substring(strDate.IndexOf("/") + 1);
                    day = int.Parse(strDate.Substring(0, strDate.IndexOf("/")));
                    strDate = strDate.Substring(strDate.IndexOf("/") + 1);
                    year = int.Parse(strDate.Substring(0, strDate.IndexOf(" ")));

                    date = new DateTime(year, month, day);

                    e.DateOfBirth = date;
                }//getting the date
                e.PhoneNumber = ds.Tables[0].Rows[0]["phoneNumber"].ToString();
                e.Email = ds.Tables[0].Rows[0]["email"].ToString();
                e.UserName = ds.Tables[0].Rows[0]["userName"].ToString();
                e.Password = ds.Tables[0].Rows[0]["password"].ToString();
                e.AddressDescription =
    ds.Tables[0].Rows[0]["addressDescription"].ToString();
            }
            return e;
        }

        //____________________Update Customer-Update_____________________

        public void UpdateAllDetails(CustomerDetails e)
        {

            strSQLName = "spUpdateCustomer";
            prmList = new ArrayList();


            prm = new OleDbParameter("@FirstName", OleDbType.VarChar);
            prm.Value = e.FirstName;
            prmList.Add(prm);

            prm = new OleDbParameter("@LastName", OleDbType.VarChar);
            prm.Value = e.LastName;
            prmList.Add(prm);

            prm = new OleDbParameter("@DateOfBirth", OleDbType.VarChar);
```

```
            prm.Value = e.DateOfBirth;
            prmList.Add(prm);

            prm = new OleDbParameter("@PhoneNumber", OleDbType.VarChar);
            prm.Value = e.PhoneNumber;
            prmList.Add(prm);

            prm = new OleDbParameter("@UserName", OleDbType.VarChar);
            prm.Value = e.UserName;
            prmList.Add(prm);

            prm = new OleDbParameter("@Email", OleDbType.VarChar);
            prm.Value = e.Email;
            prmList.Add(prm);

            prm = new OleDbParameter("@Password", OleDbType.VarChar);
            prm.Value = e.Password;
            prmList.Add(prm);

            prm = new OleDbParameter("@AddressDescription", OleDbType.VarChar);
            prm.Value = e.AddressDescription;
            prmList.Add(prm);


            prm = new OleDbParameter("@ID", OleDbType.VarChar);
            prm.Value = e.CustomerID;
            prmList.Add(prm);

            DoQueries.ExecuteSPNonQuery(strSQLName, prmList);
        }
    }


    //*************Class  Customer - end **********************
```

DoQurries.cs(Mistype)

this class is the class that contains all the functions that we want to use a lot, the ones that have to do with Access or SQL.

```
public class DoQueries
{

    private static string strConnection =
"Provider=Microsoft.ACE.OLEDB.12.0;Data Source = " +
System.Web.HttpContext.Current.Server.MapPath("~/App_Data/WorkersNetwork.accdb"
);
```

strConnection is an object of type string. It contains the path to the access database. This path is necessary for as to be able to connect to the database.

```
public DoQueries()
    {
    }
```

Constructor

Functions

```
    public static OleDbConnection Connection()
    {
        OleDbConnection cnn = new OleDbConnection(strConnection);
        return cnn;
    }
    //---------------------------
    public static OleDbDataReader ExecuteReader(string strSQLName, ArrayList
prmList)
    {
        OleDbConnection cnn = Connection();

        OleDbCommand cmd = new OleDbCommand(strSQLName, cnn);
        cmd.CommandType = CommandType.StoredProcedure;

        foreach (OleDbParameter prm in prmList)
        {
            cmd.Parameters.Add(prm);
        }

        OleDbDataReader dr = null;
        cnn.Open();
        dr = cmd.ExecuteReader();
        cnn.Close();
        return dr;
    }
    public static OleDbDataReader ExecuteReader(string strSQLName)
    {
        OleDbConnection cnn = Connection();

        OleDbCommand cmd = new OleDbCommand(strSQLName, cnn);
        cmd.CommandType = CommandType.StoredProcedure;
```

```
        OleDbDataReader dr = null;
        cnn.Open();
        dr = cmd.ExecuteReader();
        cnn.Close();
        return dr;
    }

    //---------------------------

    public static Object ExecuteSPScalar(string strSQLName, ArrayList prmList)
    {
        OleDbConnection cnn = Connection();

        OleDbCommand cmd = new OleDbCommand(strSQLName, cnn);
        cmd.CommandType = CommandType.StoredProcedure;

        foreach (OleDbParameter prm in prmList)
        {
            cmd.Parameters.Add(prm);
        }
        Object val = new Object();
        val = null;
        cnn.Open();
        val = cmd.ExecuteScalar();
        cnn.Close();

        return val;

    }

    public static Object ExecuteSPScalar(string strSQLName)
    {
        OleDbConnection cnn = Connection();

        OleDbCommand cmd = new OleDbCommand(strSQLName, cnn);
        cmd.CommandType = CommandType.StoredProcedure;

        Object val = new Object();
        val = null;

        cnn.Open();
        val = cmd.ExecuteScalar();
        cnn.Close();

        return val;

    }
    //---------------------------
    public static Object ExecuteScalar(string strSQL)
    {
        OleDbConnection cnn = Connection();
        OleDbCommand cmd = new OleDbCommand(strSQL, cnn);

        Object val = new Object();
        val = null;

        cnn.Open();
        val = cmd.ExecuteScalar();
        cnn.Close();
```

```
        return val;

    }
    //---------------------------
    public static bool ExecuteSPNonQuery(string strSQLName, ArrayList prmList)
    {
        OleDbConnection cnn = Connection();

        OleDbCommand cmd = new OleDbCommand(strSQLName, cnn);
        cmd.CommandType = CommandType.StoredProcedure;

        foreach (OleDbParameter prm in prmList)
        {
            cmd.Parameters.Add(prm);
        }
        bool r = false;

        cnn.Open();
        r = (cmd.ExecuteNonQuery() != 0);
        cnn.Close();

        return r;
    }
    public static bool ExecuteSPNonQuery(string strSQLName)
    {
        OleDbConnection cnn = Connection();

        OleDbCommand cmd = new OleDbCommand(strSQLName, cnn);
        cmd.CommandType = CommandType.StoredProcedure;

        bool r = false;

        cnn.Open();
        r = (cmd.ExecuteNonQuery() != 0);
        cnn.Close();

        return r;
    }
    //---------------------------
    public static bool ExecuteNonQuery(string strSQL)
    {
        OleDbConnection cnn = Connection();

        OleDbCommand cmd = new OleDbCommand(strSQL, cnn);
        bool r = false;

        cnn.Open();
        r = (cmd.ExecuteNonQuery() != 0);
        cnn.Close();

        return r;
    }
    //---------------------------
    public static void UpdateTableInDataBase(string strSQL, DataTable dtNew)
    {
        OleDbDataAdapter adp = new OleDbDataAdapter(strSQL, strConnection);
        OleDbCommandBuilder autoCmdBuild = new OleDbCommandBuilder(adp);

        adp.UpdateCommand = autoCmdBuild.GetUpdateCommand();
        adp.DeleteCommand = autoCmdBuild.GetDeleteCommand();
```

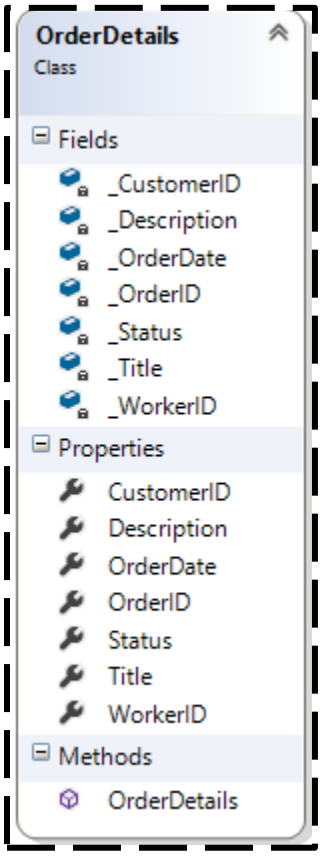
```
            adp.InsertCommand = autoCmdBuild.GetInsertCommand();
            adp.Update(dtNew);

            dtNew.Clear();
            return;
        }
        //----------------------------
        public static DataSet ExecuteDataSet(string strSQL)
        {
            DataSet ds = new DataSet();
            OleDbDataAdapter adp = new OleDbDataAdapter(strSQL, strConnection);

            adp.Fill(ds);

            return ds;
        }

        //----------------------------
        public static DataTable ExecuteDataTable(string strSQL)
        {
            DataTable dt = new DataTable();
            OleDbDataAdapter adp = new OleDbDataAdapter(strSQL, strConnection);

            adp.Fill(dt);

            return dt;
        }
         public static string GetStrConncetion()
        {
            return strConnection;
        }
    }
```

Node.cs

This class is not required for the website to work, it only helps sort data and save them temporarily in the code.

```
public class Node<T>
{
    private T x;
    private Node<T> next;

    public Node(T x)
    {
        this.x = x;
        this.next = null;
    }
    public Node(T x, Node<T> next)
    {
        this.next = next;
        this.x = x;
    }
    public T GetValue()
    {
        return x;
    }
    public Node<T> GetNext()
    {
        return next;
    }
    public bool HasNext()
    {
        if (next != null)
            return true;
        return false;
    }
    public void SetValue(T x)
    {
        this.x = x;
    }
    public void SetNext(Node<T> next)
    {
        this.next = next;
    }
    public override string ToString()
    {
        if (next != null)
            return "value:" + x + " next->" + next.ToString();
        return "value:" + x + " and thats the last node";
    }
}
```

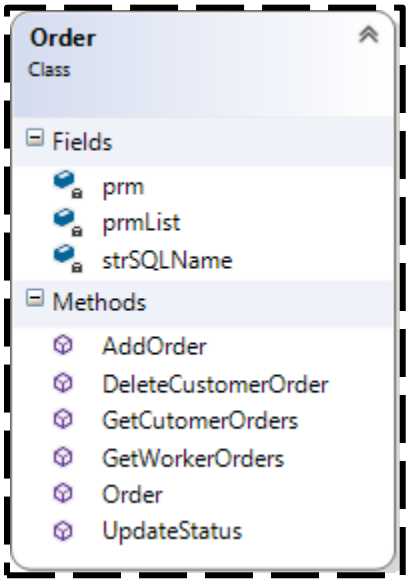
Order.cs



This class is made to contain fields.
Properties and Fields: this class has the same fields that tblOrders has. This is so we can access the order data and modify it.
Methods: there is only one method here, a constructor. This class is not made to contain methods.

```
//**************Class  OrderDetails - begin ***************
public class OrderDetails
{
    private int _OrderID;
    private string _CustomerID;
    private string _WorkerID;
    private string _Title;
    private string _Description;
    private DateTime _OrderDate;
    private string _Status;


    public OrderDetails()
    {

    }

    //Properties
    public int OrderID
    {
        get { return _OrderID; }
        set { _OrderID = value; }
    }
    public string CustomerID
```

```
        {
            get { return _CustomerID; }
            set { _CustomerID = value; }
        }
        public string WorkerID
        {
            get { return _WorkerID; }
            set { _WorkerID = value; }
        }

        public string Title
        {
            get { return _Title; }
            set { _Title = value; }
        }
        public string Description
        {
            get { return _Description; }
            set { _Description = value; }
        }
        public DateTime OrderDate
        {
            get { return _OrderDate; }
            set { _OrderDate = value; }
        }
        public string Status
        {
            get { return _Status; }
            set { _Status = value; }
        }
    }
    //**************Class  OrderDetails - end ***************
```

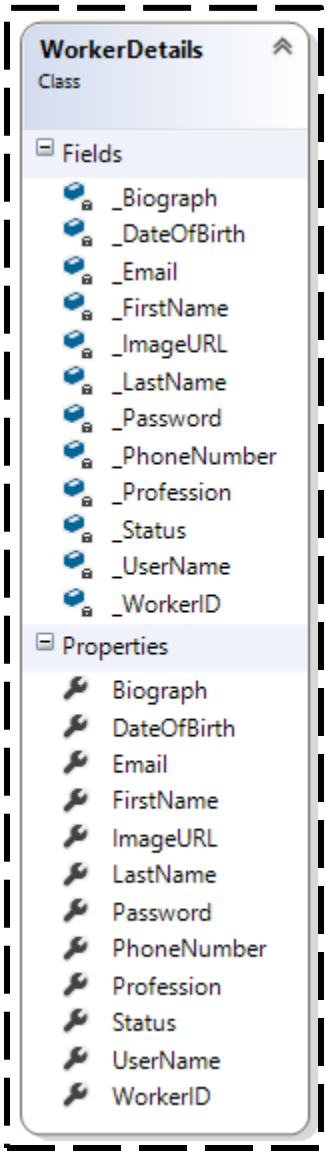


This class is made to contain methods.
Fields: the fields here are variables that we will use in methods later.
Methods:-
Order: constructor.
Add Order: adds an order using the query spAddOrder.
DeleteCustomerOrder: deletes an order by id.
GetCustomerOrders: gets all orders related to the id of the customer provided.
GetWorkerOrders: gets all orders related to the id of the worker provided.

```
//**************Class  Order - begin ********************
public class Order
{

    //Encapsulation
    ArrayList prmList;
    OleDbParameter prm;
    string strSQLName;

    //Constructor
    public Order()
    {
    }
    //Methods

    //____________________AddOrder_____________________

    public void AddOrder(OrderDetails details)
    {
        strSQLName = "spAddOrder";
        prmList = new ArrayList();

        prm = new OleDbParameter("@CustomerID", OleDbType.VarChar);
        prm.Value = details.CustomerID;
        prmList.Add(prm);

        prm = new OleDbParameter("@WorkerID", OleDbType.VarChar);
        prm.Value = details.WorkerID;
        prmList.Add(prm);

        prm = new OleDbParameter("@Title", OleDbType.VarChar);
        prm.Value = details.Title;
        prmList.Add(prm);

        prm = new OleDbParameter("@Description", OleDbType.VarChar);
        prm.Value = details.Description;
```

```
            prmList.Add(prm);


            DoQueries.ExecuteSPNonQuery(strSQLName, prmList);
        }
        public DataTable GetCutomerOrders(string id)
        {
            string strSQL = "SELECT * FROM tblOrders WHERE customerID=\"" +
                id + "\";";
            DataTable dt = DoQueries.ExecuteDataTable(strSQL);
            return dt;
        }
        public DataTable GetWorkerOrders(string id)
        {
            string strSQL = "SELECT * FROM tblOrders WHERE workerID=\"" +
                id + "\";";
            DataTable dt = DoQueries.ExecuteDataTable(strSQL);
            return dt;
        }
        public DataTable DeleteCustomerOrder(string OrderID)
        {
            string strSQL = "DELETE * " +
                "FROM tblOrders " +
                "WHERE orderID ="+OrderID+";";
            DataTable dt = DoQueries.ExecuteDataTable(strSQL);//useless
            return dt;
        }
        public DataTable UpdateStatus(string Status, string OrderID)
        {
            string strSQL = "UPDATE tblOrders SET " +
                "status= \"" + Status + "\" " +
                "WHERE orderID =" + OrderID + ";";
            DataTable dt = DoQueries.ExecuteDataTable(strSQL);//useless
            return dt;
        }
    }
    //**************Class  Order - end ********************
```

Worker.cs

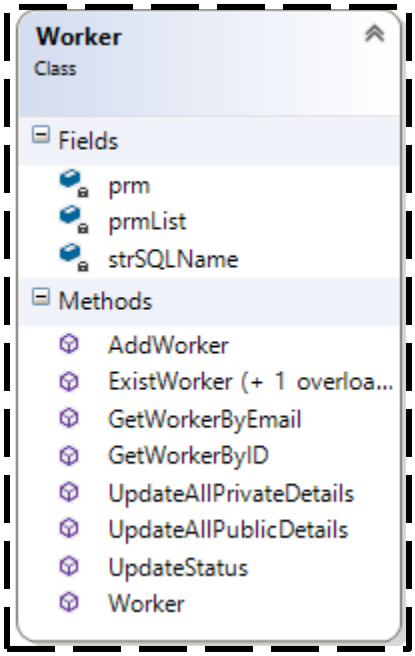


This class is made to contain fields.
Properties and Fields: this class has the same fields that tblWorkers has. This is so we can access the worker data and modify it.
Methods: there is only one method here, a constructor. This class is not made to contain methods.

```
//**************Class  WorkerDetails - begin ***************
public class WorkerDetails
{
    private string _WorkerID;
    private string _FirstName;
    private string _LastName;
    private DateTime _DateOfBirth;
    private string _PhoneNumber;
    private string _Email;
    private string _UserName;
    private string _Password;
    private string _Profession;
    private string _ImageURL;
    private string _Biograph;
    private string _Status;
```

```
//Properties
 public string WorkerID
 {
     get { return _WorkerID; }
     set { _WorkerID = value; }
 }
 public string FirstName
 {
     get { return _FirstName; }
     set { _FirstName = value; }
 }
 public string LastName
 {
     get { return _LastName; }
     set { _LastName = value; }
 }
 public DateTime DateOfBirth
 {
     get { return _DateOfBirth; }
     set { _DateOfBirth = value; }
 }
 public string PhoneNumber
 {
     get { return _PhoneNumber; }
     set { _PhoneNumber = value; }
 }
 public string Email
 {
     get { return _Email; }
     set { _Email = value; }
 }
 public string UserName
 {
     get { return _UserName; }
     set { _UserName = value; }
 }
 public string Password
 {
     get { return _Password; }
     set { _Password = value; }
 }
 public string Profession
 {
     get { return _Profession; }
     set { _Profession = value; }
 }
 public string ImageURL
 {
     get { return _ImageURL; }
     set { _ImageURL = value; }
 }
 public string Biograph
 {
     get { return _Biograph; }
     set { _Biograph = value; }
 }
 public string Status
 {
     get { return _Status; }
```

```
        set { _Status = value; }
    }
}
//**************Class  WorkerDetails - end ***************
```



This class is made to contain methods.
Fields: the fields here are variables that we will use in methods later.
Methods:-
Worker: constructor.
AddWorker: adds a worker using the query spAddWorker.
ExistWorker: checks if the worker exists using the query spExistWorker.
GetWorkerByEmail: gets the worker by email as WorkerDetails object.
GetWorkerByID: gets the worker by ID as WorkerDetails object.
UpdateAllPrivateDetails: updates the email and password for the worker using the query spUpdateWorkerPrivateInfo.
UpdateAllPublicDetails: updates all info that is shown to the public using the query spUpdateWorkerPublicInfo.
UpdateStatus: updates the status of the worker using the query spUpdateWorkerPublicInfo.

```
//**************Class  Worker - begin ********************
public class Worker
{
    //Encapsulation
    ArrayList prmList;
    OleDbParameter prm;
    string strSQLName;

    //Constructor
    public Worker()
    {
    }
    //Methods

    //____________________AddWorker_____________________

    public void AddWorker(WorkerDetails wrkrDetails)
```

```
{
    strSQLName = "spAddWorker";
    prmList = new ArrayList();

    prm = new OleDbParameter("@WorkerID", OleDbType.VarChar);
    prm.Value = wrkrDetails.WorkerID;
    prmList.Add(prm);

    prm = new OleDbParameter("@FirstName", OleDbType.VarChar);
    prm.Value = wrkrDetails.FirstName;
    prmList.Add(prm);

    prm = new OleDbParameter("@LastName", OleDbType.VarChar);
    prm.Value = wrkrDetails.LastName;
    prmList.Add(prm);

    prm = new OleDbParameter("@DateOfBirth", OleDbType.VarChar);
    prm.Value = wrkrDetails.DateOfBirth;
    prmList.Add(prm);

    prm = new OleDbParameter("@PhoneNumber", OleDbType.VarChar);
    prm.Value = wrkrDetails.PhoneNumber;
    prmList.Add(prm);

    prm = new OleDbParameter("@Email", OleDbType.VarChar);
    prm.Value = wrkrDetails.Email;
    prmList.Add(prm);

    prm = new OleDbParameter("@UserName", OleDbType.VarChar);
    prm.Value = wrkrDetails.UserName;
    prmList.Add(prm);

    prm = new OleDbParameter("@Password", OleDbType.VarChar);
    prm.Value = wrkrDetails.Password;
    prmList.Add(prm);

    prm = new OleDbParameter("@Profession", OleDbType.VarChar);
    prm.Value = wrkrDetails.Profession;
    prmList.Add(prm);


    DoQueries.ExecuteSPNonQuery(strSQLName, prmList);
}

//____________________ExistWorker_____________________________
public Boolean ExistWorker(string Email, bool useless)
{
    strSQLName = "spExistWorker";
    prmList = new ArrayList();

    prm = new OleDbParameter("@Email", OleDbType.VarChar);
    prm.Value = Email;
    prmList.Add(prm);

    prm = new OleDbParameter("@WorkerID", OleDbType.VarChar);
    prm.Value = "a-1-8-8-8-7-9";
    prmList.Add(prm);

    int intCount = (int)DoQueries.ExecuteSPScalar(strSQLName, prmList);
    return intCount > 0;
```

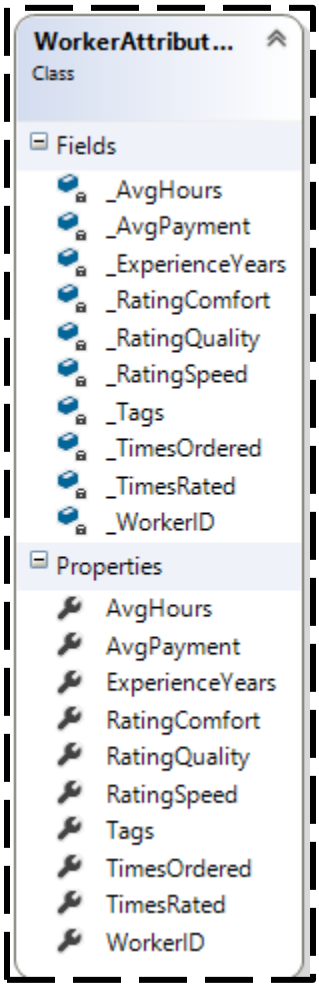
```
        }
        public Boolean ExistWorker(string ID)
        {
            strSQLName = "spExistWorker";
            prmList = new ArrayList();

            prm = new OleDbParameter("@Email", OleDbType.VarChar);
            prm.Value = "a-1-8-8-8-7-9";
            prmList.Add(prm);

            prm = new OleDbParameter("@WorkerID", OleDbType.VarChar);
            prm.Value = ID;
            prmList.Add(prm);

            int intCount = (int)DoQueries.ExecuteSPScalar(strSQLName, prmList);
            return intCount > 0;
        }

        [WebMethod]
        public WorkerDetails GetWorkerByID(string id)
        {
            WorkerDetails e = new WorkerDetails();

            if (ExistWorker(id))
            {
                string strDetails =
                    string.Format("SELECT * FROM tblWorkers" +
                    " WHERE workerID='{0}'", id);
                DataSet ds = DoQueries.ExecuteDataSet(strDetails);
                e.WorkerID = ds.Tables[0].Rows[0]["workerID"].ToString();
                e.FirstName = ds.Tables[0].Rows[0]["firstName"].ToString();
                e.LastName = ds.Tables[0].Rows[0]["lastName"].ToString();
                {
                    int day, month, year;
                    DateTime date;
                    string strDate =
ds.Tables[0].Rows[0]["dateOfBirth"].ToString();

                    month = int.Parse(strDate.Substring(0, strDate.IndexOf("/")));
                    strDate = strDate.Substring(strDate.IndexOf("/") + 1);
                    day = int.Parse(strDate.Substring(0, strDate.IndexOf("/")));
                    strDate = strDate.Substring(strDate.IndexOf("/") + 1);
                    year = int.Parse(strDate.Substring(0, strDate.IndexOf(" ")));

                    date = new DateTime(year, month, day);

                    e.DateOfBirth = date;
                }//getting the date
                e.PhoneNumber = ds.Tables[0].Rows[0]["phoneNumber"].ToString();
                e.Email = ds.Tables[0].Rows[0]["email"].ToString();
                e.UserName = ds.Tables[0].Rows[0]["userName"].ToString();
                e.Password = ds.Tables[0].Rows[0]["password"].ToString();
                e.Profession = ds.Tables[0].Rows[0]["profession"].ToString();
                e.ImageURL = ds.Tables[0].Rows[0]["imageURL"].ToString();
                e.Biograph = ds.Tables[0].Rows[0]["biograph"].ToString();
                e.Status = ds.Tables[0].Rows[0]["status"].ToString();
            }
            return e;
        }
```

```
        public WorkerDetails GetWorkerByEmail(string email)
        {
            WorkerDetails e = new WorkerDetails();

            if (ExistWorker(email, true))
            {
                string strDetails =
                    string.Format("SELECT * FROM tblWorkers" +
                    " WHERE email='{0}'", email);
                DataSet ds = DoQueries.ExecuteDataSet(strDetails);
                e.WorkerID = ds.Tables[0].Rows[0]["workerID"].ToString();
                e.FirstName = ds.Tables[0].Rows[0]["firstName"].ToString();
                e.LastName = ds.Tables[0].Rows[0]["lastName"].ToString();
                {
                    int day, month, year;
                    DateTime date;
                    string strDate =
    ds.Tables[0].Rows[0]["dateOfBirth"].ToString();

                    month = int.Parse(strDate.Substring(0, strDate.IndexOf("/")));
                    strDate = strDate.Substring(strDate.IndexOf("/") + 1);
                    day = int.Parse(strDate.Substring(0, strDate.IndexOf("/")));
                    strDate = strDate.Substring(strDate.IndexOf("/") + 1);
                    year = int.Parse(strDate.Substring(0, strDate.IndexOf(" ")));

                    date = new DateTime(year, month, day);

                    e.DateOfBirth = date;
                }//getting the date
                e.PhoneNumber = ds.Tables[0].Rows[0]["phoneNumber"].ToString();
                e.Email = email;
                e.UserName = ds.Tables[0].Rows[0]["userName"].ToString();
                e.Password = ds.Tables[0].Rows[0]["password"].ToString();
                e.Profession = ds.Tables[0].Rows[0]["profession"].ToString();
                e.ImageURL = ds.Tables[0].Rows[0]["imageURL"].ToString();
                e.Biograph = ds.Tables[0].Rows[0]["biograph"].ToString();
                e.Status = ds.Tables[0].Rows[0]["status"].ToString();
            }
            return e;
        }

        //____________________Update Worker-Update_____________________
        public void UpdateAllPublicDetails(WorkerDetails e)
        {

            strSQLName = "spUpdateWorkerPublicInfo";
            prmList = new ArrayList();


            prm = new OleDbParameter("@FirstName", OleDbType.VarChar);
            prm.Value = e.FirstName;
            prmList.Add(prm);

            prm = new OleDbParameter("@LastName", OleDbType.VarChar);
            prm.Value = e.LastName;
            prmList.Add(prm);

            prm = new OleDbParameter("@DateOfBirth", OleDbType.VarChar);
            prm.Value = e.DateOfBirth;
            prmList.Add(prm);
```

```
        prm = new OleDbParameter("@PhoneNumber", OleDbType.VarChar);
        prm.Value = e.PhoneNumber;
        prmList.Add(prm);

        prm = new OleDbParameter("@UserName", OleDbType.VarChar);
        prm.Value = e.UserName;
        prmList.Add(prm);

        prm = new OleDbParameter("@ImageURL", OleDbType.VarChar);
        prm.Value = e.ImageURL;
        prmList.Add(prm);

        prm = new OleDbParameter("@Biograph", OleDbType.VarChar);
        prm.Value = e.Biograph;
        prmList.Add(prm);

        prm = new OleDbParameter("@Status", OleDbType.VarChar);
        prm.Value = e.Status;
        prmList.Add(prm);

        prm = new OleDbParameter("@ID", OleDbType.VarChar);
        prm.Value = e.WorkerID;
        prmList.Add(prm);

        DoQueries.ExecuteSPNonQuery(strSQLName, prmList);
    }

    public void UpdateAllPrivateDetails(WorkerDetails e)
    {
        strSQLName = "spUpdateWorkerPrivateInfo";
        prmList = new ArrayList();

        prm = new OleDbParameter("@Email", OleDbType.VarChar);
        prm.Value = e.Email;
        prmList.Add(prm);

        prm = new OleDbParameter("@Password", OleDbType.VarChar);
        prm.Value = e.Password;
        prmList.Add(prm);

        prm = new OleDbParameter("@ID", OleDbType.VarChar);
        prm.Value = e.WorkerID;
        prmList.Add(prm);

        DoQueries.ExecuteSPNonQuery(strSQLName, prmList);
    }
    public void UpdateStatus(string id, string Status)
    {
        Worker = new Worker();
        WorkerDetails e = worker.GetWorkerByID(id);

        strSQLName = "spUpdateWorkerPublicInfo";
        prmList = new ArrayList();


        prm = new OleDbParameter("@FirstName", OleDbType.VarChar);
        prm.Value = e.FirstName;
        prmList.Add(prm);
```

```
            prm = new OleDbParameter("@LastName", OleDbType.VarChar);
            prm.Value = e.LastName;
            prmList.Add(prm);

            prm = new OleDbParameter("@DateOfBirth", OleDbType.VarChar);
            prm.Value = e.DateOfBirth;
            prmList.Add(prm);

            prm = new OleDbParameter("@PhoneNumber", OleDbType.VarChar);
            prm.Value = e.PhoneNumber;
            prmList.Add(prm);

            prm = new OleDbParameter("@UserName", OleDbType.VarChar);
            prm.Value = e.UserName;
            prmList.Add(prm);

            prm = new OleDbParameter("@ImageURL", OleDbType.VarChar);
            prm.Value = e.ImageURL;
            prmList.Add(prm);

            prm = new OleDbParameter("@Biograph", OleDbType.VarChar);
            prm.Value = e.Biograph;
            prmList.Add(prm);

            prm = new OleDbParameter("@Status", OleDbType.VarChar);
            prm.Value = Status;
            prmList.Add(prm);

            prm = new OleDbParameter("@ID", OleDbType.VarChar);
            prm.Value = id;
            prmList.Add(prm);

            DoQueries.ExecuteSPNonQuery(strSQLName, prmList);
        }

    }
    //**************Class  Worker - end **********************
```

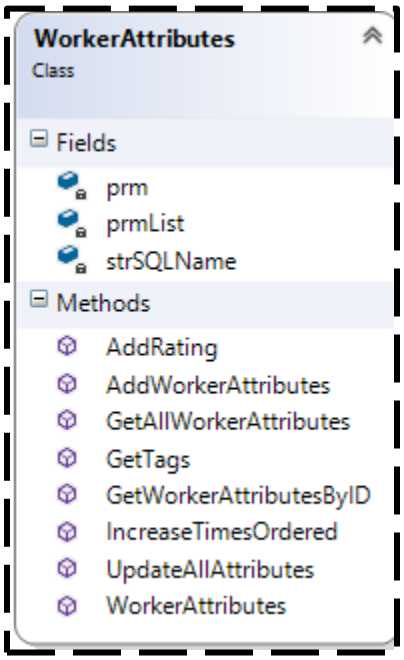
WorkerAttributes.cs



This class is made to contain fields. Properties and Fields: this class has the same fields that tblWorkerAttributes has. This is so we can access the workerAttributes data and modify it. Methods: there is only one method here, a constructor. This class is not made to contain methods.

```
//**************Class  WorkerAttributesDetails - begin **************
public class WorkerAttributesDetails
{
    private string _WorkerID;
    private double _AvgPayment;
    private double _AvgHours;
    private int _TimesOrdered;
    private double _ExperienceYears;
    private double _RatingQuality;
    private double _RatingSpeed;
    private double _RatingComfort;
    private int _TimesRated;
    private string _Tags;


    //Properties
    public string WorkerID
    {
        get { return _WorkerID; }
```

```
            set { _WorkerID = value; }
        }
        public double AvgPayment
        {
            get { return _AvgPayment; }
            set { _AvgPayment = value; }
        }
        public double AvgHours
        {
            get { return _AvgHours; }
            set { _AvgHours = value; }
        }
        public int TimesOrdered
        {
            get { return _TimesOrdered; }
            set { _TimesOrdered = value; }
        }
        public double ExperienceYears
        {
            get { return _ExperienceYears; }
            set { _ExperienceYears = value; }
        }
        public double RatingQuality
        {
            get { return _RatingQuality; }
            set { _RatingQuality = value; }
        }
        public double RatingSpeed
        {
            get { return _RatingSpeed; }
            set { _RatingSpeed = value; }
        }
        public double RatingComfort
        {
            get { return _RatingComfort; }
            set { _RatingComfort = value; }
        }
        public int TimesRated
        {
            get { return _TimesRated; }
            set { _TimesRated = value; }
        }
        public string Tags
        {
            get { return _Tags; }
            set { _Tags = value; }
        }


    }
    //**************Class  WorkerAttributesDetails - end **************
```



This class is made to contain methods.
Fields: the fields here are variables that we will use in methods later.
Methods:-
WorkerAttributes: constructor.
AddWorkerAttributes: adds workerAttributes using the query spAddWorkerAttributes.
AddRating: adds one rating to all rating fields in this table using the query spAddRating.
GetAllWorkerAttributes: gets all workerAttributes as a DataTable. Uses the query spWorkerAttributesCount.
GetTags: outputs the tags as nodes based on the string of tags provided.
GetWorkerAttributesByID: gets the worker by ID as WorkerDetails object.
IncreaseTimesOrdered: increases "timesOrdered" field value by 1 based on the id provided.
UpdateAllAttributes: updates all workerAttributes data based on id and what's

```
//**************Class  WorkerAttributes - begin ********************
public class WorkerAttributes
{
    //Encapsulation
    ArrayList prmList;
    OleDbParameter prm;
    string strSQLName;

    //Constructor
    public WorkerAttributes()
    {
    }
    //Methods

    //____________________AddWorkerAttributes_____________________

    public void AddWorkerAttributes(WorkerAttributesDetails wrkrAtrDetails)
    {
        strSQLName = "spAddWorkerAttributes";
        prmList = new ArrayList();
```

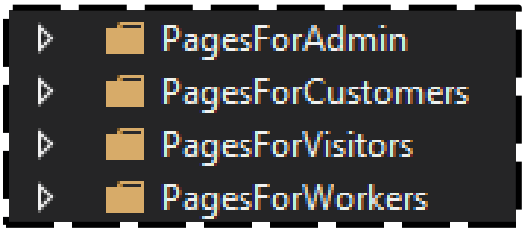
```
            prm = new OleDbParameter("@WorkerID", OleDbType.VarChar);
            prm.Value = wrkrAtrDetails.WorkerID;
            prmList.Add(prm);

            prm = new OleDbParameter("@Tags", OleDbType.VarChar);
            prm.Value = wrkrAtrDetails.Tags;
            prmList.Add(prm);

            DoQueries.ExecuteSPNonQuery(strSQLName, prmList);
        }


        [WebMethod]
        public WorkerAttributesDetails GetWorkerAttributesByID(string id)
        {
            WorkerAttributesDetails e = new WorkerAttributesDetails();

            string strDetails =
                string.Format("SELECT * FROM tblWorkerAttributes" +
                " WHERE workerID='{0}'", id);
            DataSet ds = DoQueries.ExecuteDataSet(strDetails);
            e.WorkerID = ds.Tables[0].Rows[0]["workerID"].ToString();
            e.AvgPayment =
    double.Parse(ds.Tables[0].Rows[0]["avgPayment"].ToString());
            e.AvgHours = double.Parse(ds.Tables[0].Rows[0]["avgHours"].ToString());
            e.TimesOrdered =
    int.Parse(ds.Tables[0].Rows[0]["timesOrdered"].ToString());
            e.ExperienceYears =
    double.Parse(ds.Tables[0].Rows[0]["experienceYears"].ToString());
            e.RatingQuality =
    double.Parse(ds.Tables[0].Rows[0]["ratingQuality"].ToString());
            e.RatingSpeed =
    double.Parse(ds.Tables[0].Rows[0]["ratingSpeed"].ToString());
            e.RatingComfort =
    double.Parse(ds.Tables[0].Rows[0]["ratingComfort"].ToString());
            e.TimesRated =
    int.Parse(ds.Tables[0].Rows[0]["timesRated"].ToString());
            e.Tags = ds.Tables[0].Rows[0]["tags"].ToString();

            return e;
        }


        [WebMethod]
        public Node<WorkerAttributesDetails> GetAllWorkerAttributes()
        {
            Node<WorkerAttributesDetails> all, temp;
            WorkerAttributesDetails e = new WorkerAttributesDetails();

            string strDetails =
                   string.Format("SELECT * FROM tblWorkerAttributes");
            DataSet ds = DoQueries.ExecuteDataSet(strDetails);
            e.WorkerID = ds.Tables[0].Rows[0]["workerID"].ToString();
            e.AvgPayment =
    double.Parse(ds.Tables[0].Rows[0]["avgPayment"].ToString());
            e.AvgHours = double.Parse(ds.Tables[0].Rows[0]["avgHours"].ToString());
            e.TimesOrdered =
    int.Parse(ds.Tables[0].Rows[0]["timesOrdered"].ToString());
            e.ExperienceYears =
    double.Parse(ds.Tables[0].Rows[0]["experienceYears"].ToString());
```

```
            e.RatingQuality =
double.Parse(ds.Tables[0].Rows[0]["ratingQuality"].ToString());
            e.RatingSpeed =
double.Parse(ds.Tables[0].Rows[0]["ratingSpeed"].ToString());
            e.RatingComfort =
double.Parse(ds.Tables[0].Rows[0]["ratingComfort"].ToString());
            e.TimesRated =
int.Parse(ds.Tables[0].Rows[0]["timesRated"].ToString());
            e.Tags = ds.Tables[0].Rows[0]["tags"].ToString();

            all = new Node<WorkerAttributesDetails>(e);
            temp = all;
            int count = (int)DoQueries.ExecuteSPScalar("spWorkerAttributesCount");

            for (int i = 1; i < count; i++)
            {
                e = new WorkerAttributesDetails();
                e.WorkerID = ds.Tables[0].Rows[i]["workerID"].ToString();
                e.AvgPayment =
double.Parse(ds.Tables[0].Rows[i]["avgPayment"].ToString());
                e.AvgHours =
double.Parse(ds.Tables[0].Rows[i]["avgHours"].ToString());
                e.TimesOrdered =
int.Parse(ds.Tables[0].Rows[i]["timesOrdered"].ToString());
                e.ExperienceYears =
int.Parse(ds.Tables[0].Rows[i]["experienceYears"].ToString());
                e.RatingQuality =
double.Parse(ds.Tables[0].Rows[i]["ratingQuality"].ToString());
                e.RatingSpeed =
double.Parse(ds.Tables[0].Rows[i]["ratingSpeed"].ToString());
                e.RatingComfort =
double.Parse(ds.Tables[0].Rows[i]["ratingComfort"].ToString());
                e.TimesRated =
int.Parse(ds.Tables[0].Rows[i]["timesRated"].ToString());
                e.Tags = ds.Tables[0].Rows[i]["tags"].ToString();

                temp.SetNext(new Node<WorkerAttributesDetails>(e));
                temp = temp.GetNext();
            }


            return all;
        }

        public Node<string> GetTags(string str)
        {
            Node<string> result, p;

            //starts splitting the tags into nodes

            str = str.Substring(1);
            if (str.Contains("#"))
            {
                result = new Node<string>("#" + str.Substring(0,
str.IndexOf("#")));
                p = result;
                str = str.Substring(str.IndexOf("#") + 1);
                while (str != "")
                {
                    if (str.Contains("#"))
```

```
                {
                    p.SetNext(new Node<string>("#" + str.Substring(0,
str.IndexOf("#"))));
                    str = str.Substring(str.IndexOf("#") + 1);
                }
                else
                {
                    p.SetNext(new Node<string>("#" + str));
                    str = "";
                }
                p = p.GetNext();
            }
        }
        else
            result = new Node<string>("#" + str);
        return result;
    }

    //____________________AddRating-Update_____________________
    public void AddRating(string id, double ratingQuality, double ratingSpeed,
double ratingComfort)
    {
        WorkerAttributes = new WorkerAttributes();
        WorkerAttributesDetails e =
workerAttributes.GetWorkerAttributesByID(id);

        ratingQuality = (e.RatingQuality * e.TimesRated + ratingQuality) /
(e.TimesRated+1);
        ratingSpeed = (e.RatingSpeed * e.TimesRated + ratingSpeed) /
(e.TimesRated + 1);
        ratingComfort = (e.RatingComfort * e.TimesRated + ratingComfort) /
(e.TimesRated + 1);
        e.TimesRated++;

        strSQLName = "spAddRating";
        prmList = new ArrayList();

        prm = new OleDbParameter("@RatingQuality", OleDbType.VarChar);
        prm.Value = ratingQuality;
        prmList.Add(prm);

        prm = new OleDbParameter("@RatingSpeed", OleDbType.VarChar);
        prm.Value = ratingSpeed;
        prmList.Add(prm);

        prm = new OleDbParameter("@RatingComfort", OleDbType.VarChar);
        prm.Value = ratingComfort;
        prmList.Add(prm);

        prm = new OleDbParameter("@TimesRated", OleDbType.VarChar);
        prm.Value = e.TimesRated;
        prmList.Add(prm);

        prm = new OleDbParameter("@ID", OleDbType.VarChar);
        prm.Value = id;
        prmList.Add(prm);

        DoQueries.ExecuteSPNonQuery(strSQLName, prmList);
    }
    //____________________Update worker attributes-Update_____________________
```

```
        public void UpdateAllAttributes(WorkerAttributesDetails e)
        {
            strSQLName = "spUpdateWorkerAttributes";
            prmList = new ArrayList();


            prm = new OleDbParameter("@AvgPayment", OleDbType.VarChar);
            prm.Value = e.AvgPayment;
            prmList.Add(prm);

            prm = new OleDbParameter("@AvgHours", OleDbType.VarChar);
            prm.Value = e.AvgHours;
            prmList.Add(prm);

            prm = new OleDbParameter("@ExperienceYears", OleDbType.VarChar);
            prm.Value = e.ExperienceYears;
            prmList.Add(prm);

            prm = new OleDbParameter("@Tags", OleDbType.VarChar);
            prm.Value = e.Tags;
            prmList.Add(prm);

            prm = new OleDbParameter("@TimesOrdered", OleDbType.VarChar);
            prm.Value = e.TimesOrdered;
            prmList.Add(prm);

            prm = new OleDbParameter("@ID", OleDbType.VarChar);
            prm.Value = e.WorkerID;
            prmList.Add(prm);

            DoQueries.ExecuteSPNonQuery(strSQLName, prmList);
        }
        public void IncreaseTimesOrdered(string id)
        {
            WorkerAttributes worker = new WorkerAttributes();
            int Times=worker.GetWorkerAttributesByID(id).TimesOrdered+1;
                string strSQL = "UPDATE tblWorkerAttributes SET " +
                "timesOrdered= " + Times +" " +
                "WHERE workerID =\"" + id + "\";";
            DataTable dt = DoQueries.ExecuteDataTable(strSQL);//useless
        }

    }
      //**************Class  WorkerAttributes - end **********************
```

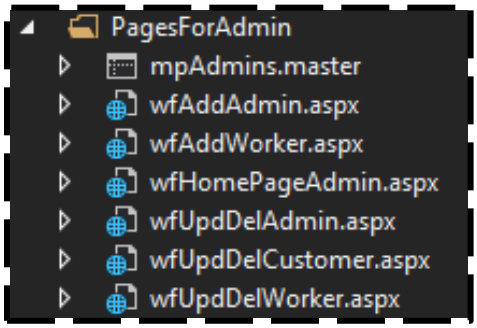
User Interface



These folders contain all the pages that users can access.
They are split that way to restrict the users from accessing each other's pages, and to keep the information more secretive.

As stated before, every webform has a MasterPage.
The MasterPage for each webform has been decided according to the user, if the user of the webform is an admin, the webform's master would be mpAdmins.master, and so on…
The master files contain a header, this header is a Web-User-Control, the control's name is decided based on the target user. Example: wucCustomer…

PagesForAdmin



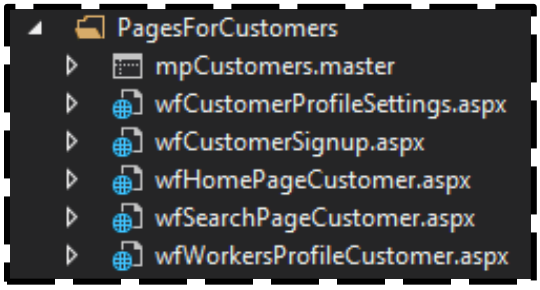
mpAdmins.master:

Webforms:

- wfHomePageAdmin: the homepage for the admins. Contains instructions on how to use the website as an admin.
- wfAddAdmin: a page that allows an admin to add an admin.
- wfAddWorker: a page that allows an admin to add a worker.

- wfUpdDelAdmin: a page that allows an admin to Update or Delete admins from the database.
- wfUpdDelCustomer: a page that allows an admin to Update or Delete customers from the database.
- wfUpdDelWorker: a page that allows an admin to Update or Delete workers from the database.
- wfUpdDelWorkerAttributes: a page that allows an admin to Update or Delete workerAttributes from the database.

PagesForCustomers



mpCustomers.master:

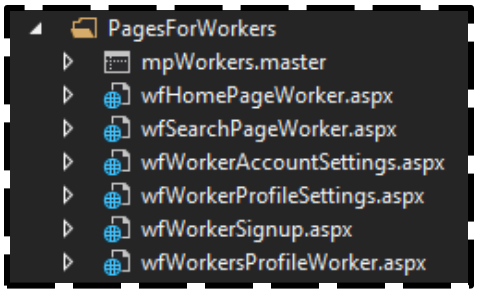
Webforms:

- wfHomePageCustomer: the homepage for the customers. Contains instructions on how to use the website as a customer.
- wfCustomerProfileSettings: allows the customer to change/set their profile information.
- wfCustomerSignup: a sign-up page for customers.

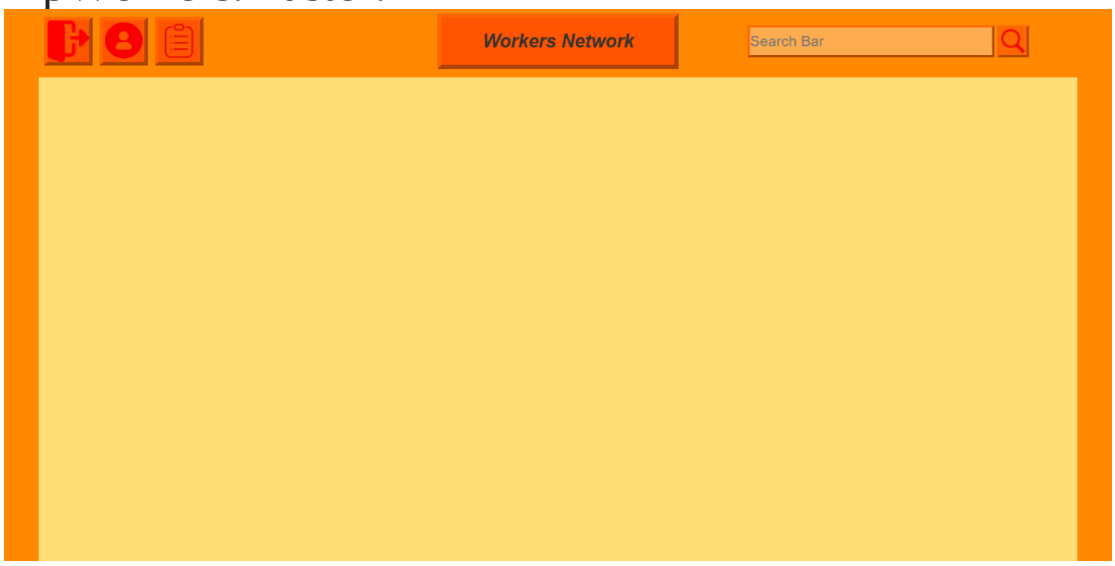
- wfSearchPageCustomer: the page of the results the customer gets when they search for workers. Shows a brief summary about every worker In the results.

- wfWorkersProfileCustomer: customers enter this page by pressing the search result that suits them best. This page includes all the important info about the worker - the info provided by them. Customers can make orders and rate workers from this page.

PagesForWorkers



mpWorkers.master:



Webforms:

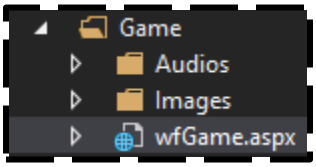
- wfHomePageWorker: the homepage for the workers. Contains instructions on how to use the website as a worker.
- wfWorkerProfileSettings: allows the worker to change/set their profile information as shown to the public.
- wfWorkerAccountSettings: allows the worker to change their account settings/info, like their email address, or to change their password.
- wfWorkerSignup: a sign-up page for workers.

- wfSearchPageCustomer: the page of the results the worker gets when they search for other workers. Results are shown like the above example.
  The search feature exists for workers too. It exists to allow workers to search for their competition.
- wfWorkersProfileCustomer: workers enter this page by pressing the search result that suits them best. This page includes all the important info about the worker - the info provided by them. Workers can NOT make orders and rate workers from this page.

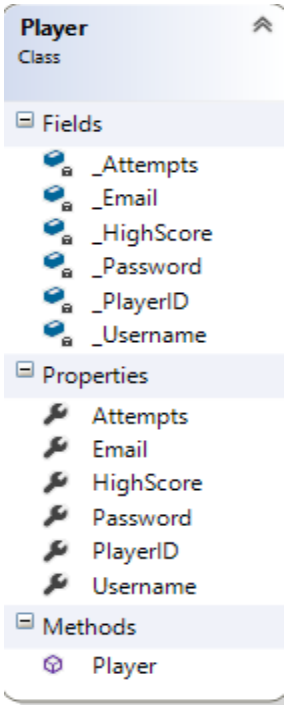
PagesForVisitors

There is no master page because this folder includes only two webforms, and they both have different designs.
wfLogIn: in this webform, users enter their email address and their password to log in.
wfStartingPage: this is the starting page. In this page you get to choose if you want to use the website as a worker or a customer. Admins choose to use this website as a worker. Admins cant have identical emails with workers, because they log-in as workers.

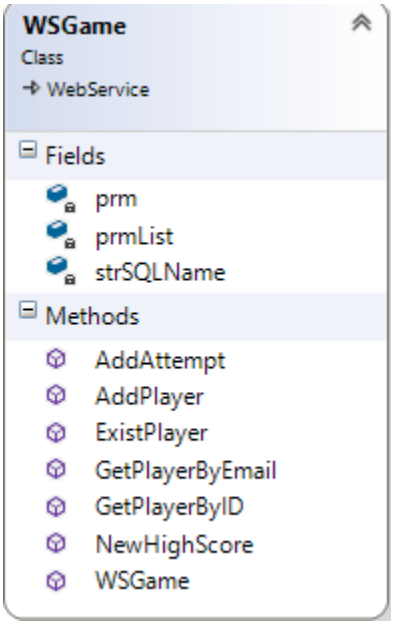
Web Service



This is where I use the web service in my website



This class is made to contain fields. Properties and Fields: this class has the same fields that tblPlayer has in the WebService database. This is so we can access the player data and modify it. Methods: there is only one method here, a constructor. This class is not made to contain methods.

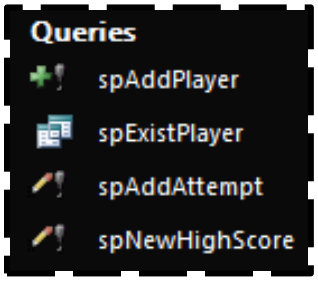


This class is made to contain methods.
Fields: the fields here are variables that we will use in methods later.
Methods:-
AddPlayer: adds a player to the data base using the query spAddPlayer.
AddPlayer: adds an attempt to the player using the query spAddAttempt.
ExistPlayer: checks if a player with the same email already exists using the query spExistPlayer.
GetPlayerByEmail: gets a player by their email.
GetPlayerByID: gets a player by their ID
NewHighScore: updates the highScore field to the player.
WSGame: constructor.

tblPlayer:

playerID	AutoNumber	A unique serial number for the player.
email	Short Text	the player's email address.
username	Short Text	the player's username.
password	Short Text	the player's password to their account.
highScore	Number	the highest score the player got.
attempts	Number	the number of attempts the player made.

Web service queries:



spAddPlayer

```
INSERT INTO tblPlayer ( email, username, [password] )
VALUES ([@Email], [@Username], [@Password]);
```

The use of this query is to add a player. It is activated when a new player joins.

spExistPlayer

```
SELECT count(*) FROM tblPlayer
WHERE email=[@Email];
```

The use of this query is to count the number of players that exist with the same email that is provided.
It is essentially used to check if a player with the email provided already exists or not, to avoid duplicates.

spAddAttempt

UPDATE tblPlayer SET attempts = [@Attempts]
WHERE playerID=[@PlayerID];

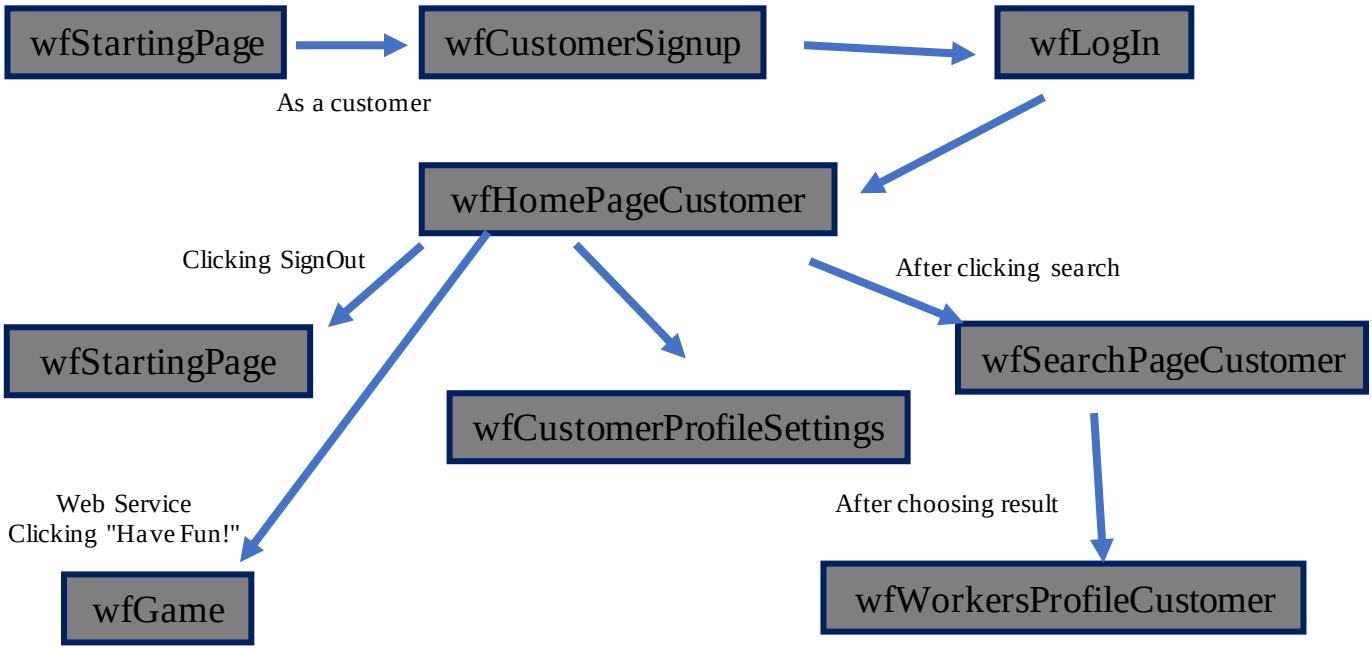
The use of this query is to add an attempt. It is used when a player uses an attempt.

spNewHighScore

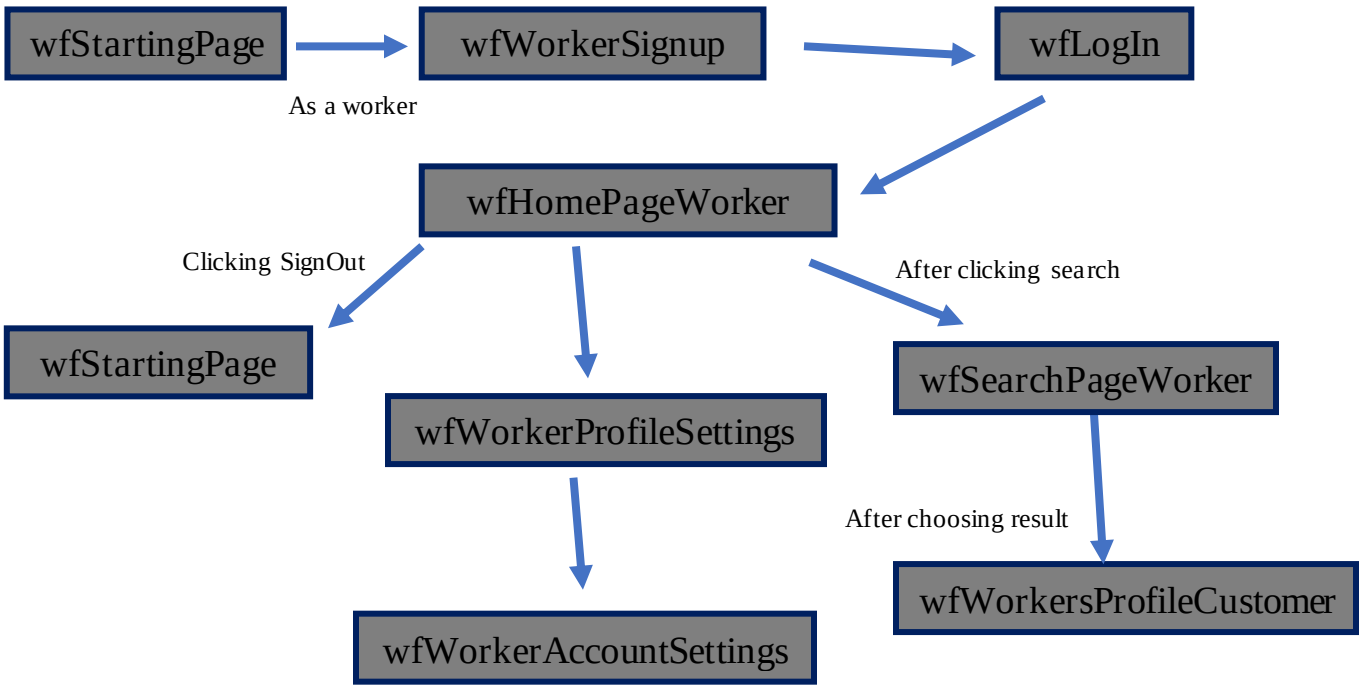
UUPDATE tblPlayer SET highScore = [@HighScore]
WHERE playerID=[@PlayerID];

The use of this query is to update the highScore value of a player. It is used when a player gets a new high score in the game.

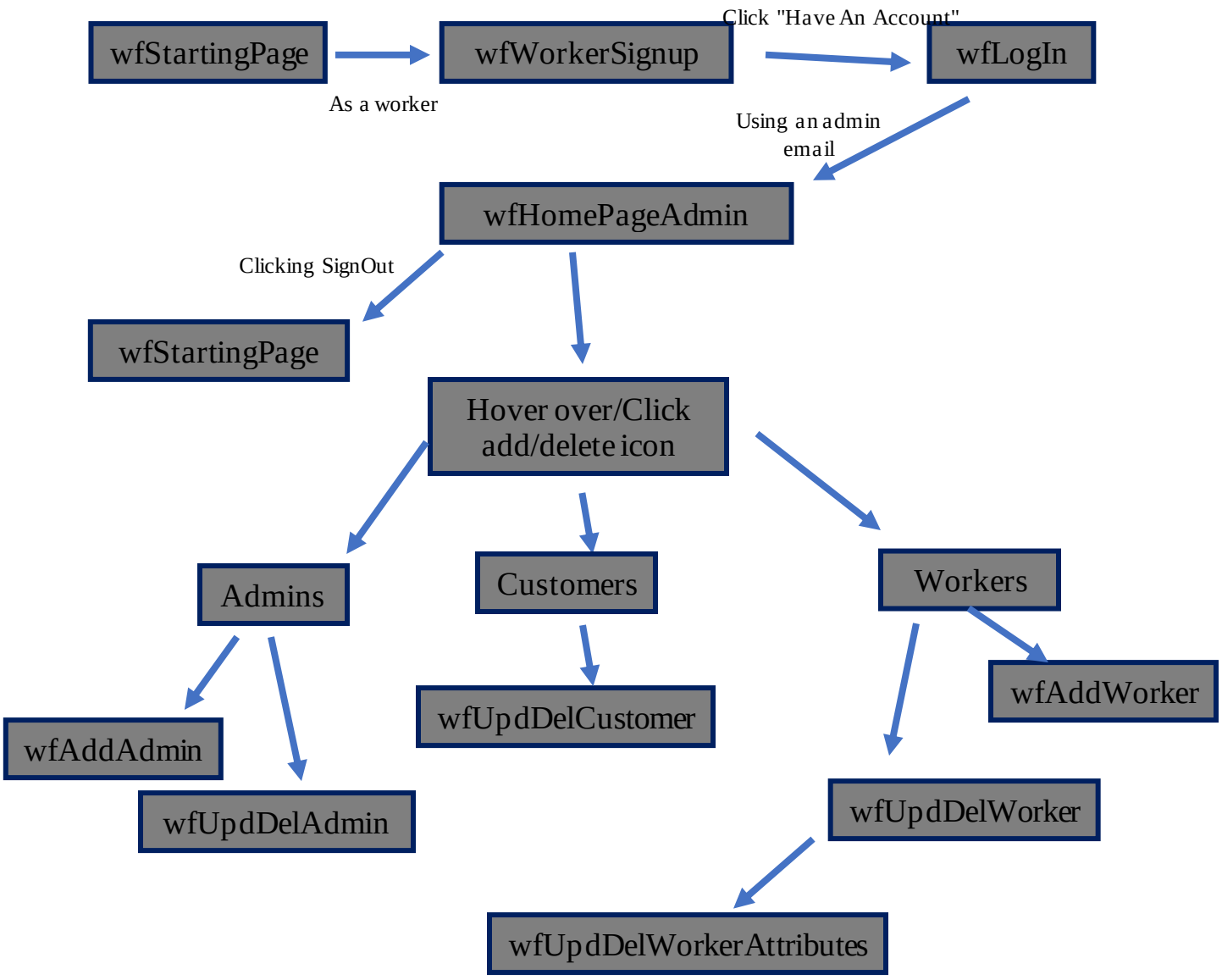
Customers' website map



Workers' website map



Admins' website map



Accounts to use for testing the project:

Admin:

- Email: admin@gmail.com.
- Password: 987.

Worker:

- Email: worker@gmail.com.
- Password: 654.

Customer:

- Email: customer@gmail.com.
- Password: 321.

My opinion about this project

I consider this project to be my biggest project yet.

I learned a lot of things from it, from understanding about work in our society, to learning new code languages.

I took it upon myself in the past six months to learn most things about CSS, Java Script, and HTML. I decided to learn them although I did not have to do that for this project for a lot of reasons…

The reasons being:

1. I was not content or happy with the appearance of my project without using CSS or HTML.
2. I wanted to learn new code languages because I was sure that I will need them in the future. Also to better my coding skills.

I used CSS and HTML for appearance. So I did my research and browsed through the source code of a lot of big websites, like YouTube or Instagram. I learned from them a lot. I also searched a lot on the internet for solutions to my problems, or to learn how to do things in CSS.

I used JavaScript a lot less than the other two. I mostly used it to avoid PostBack if I could. I also

used it when it was much easier to do what I wanted in JS rather than C#.

It was tough to do this project for me because I was always ahead of my class, and couldn’t ask anyone about it. But I liked the challenge and I liked working on it.